

# **Sissejuhatus infotehnoloogiasse**

**Tarkvara arhitektuur**

- Tarkvara arhitektuur suuremas pildis
- Olulisi punkte arusaamiseks
- Süsteemide komponendid
- Süsteemide sidumine
- Teistest arusaamine
- Tarkvara litsentsid ja vabatarkvara litsentside näited

# Tarkvara arhitektuur

---

- Arhitektuuri all mõeldakse IT-s:
  - mingi süsteemi tehnoloogilisi põhimõtteid
  - millisteks suurteks osadeks süsteem jaotub
  - kuidas osad omavahel suhtlevad
  - milliseid suuri valmistükke süsteem kasutab
  
- Tarkvaras on sellisteks küsimusteks näiteks:
  - Mis opsüsteemi alla rakendus teha
  - Kas hoida infot lihtsalt failides või andmebaasis
  - Millist veebiserverit ja kuidas täpselt kasutada
  - Kas pöörduda andmebaasi poole otse või csv või XML või json vahekihi abil
  - Mis keeles/keeltes rakendus teha
  - Millist kasutajaliidese teeki kasutada
  - ... jne ....

Tarkvara arhitektuur suuremas pildis

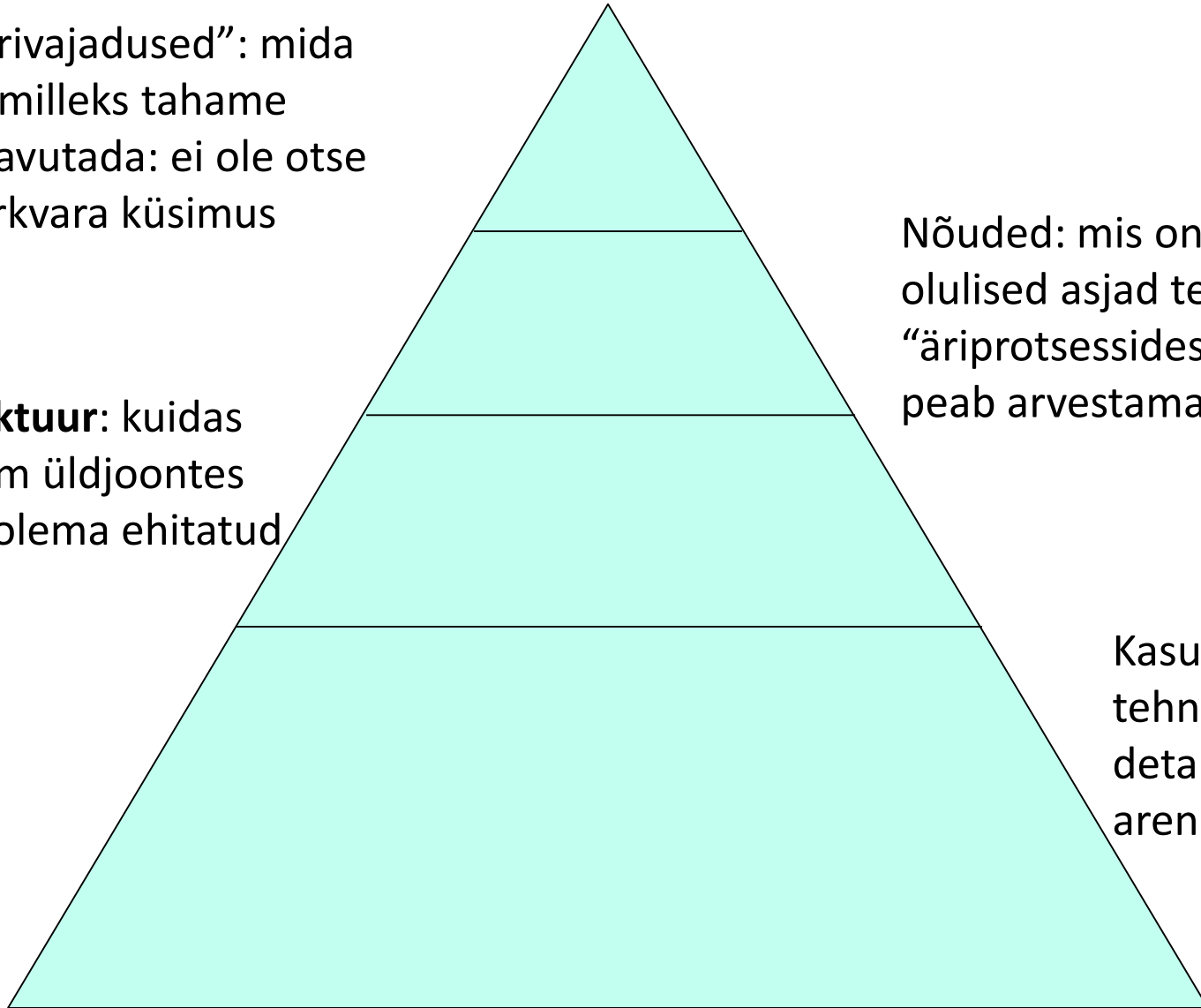
# Suurem pilt: milleks me süsteemi ehitame?

“Ärivateadused”: mida  
ja milleks tahame  
saavutada: ei ole otse  
tarkvara küsimus

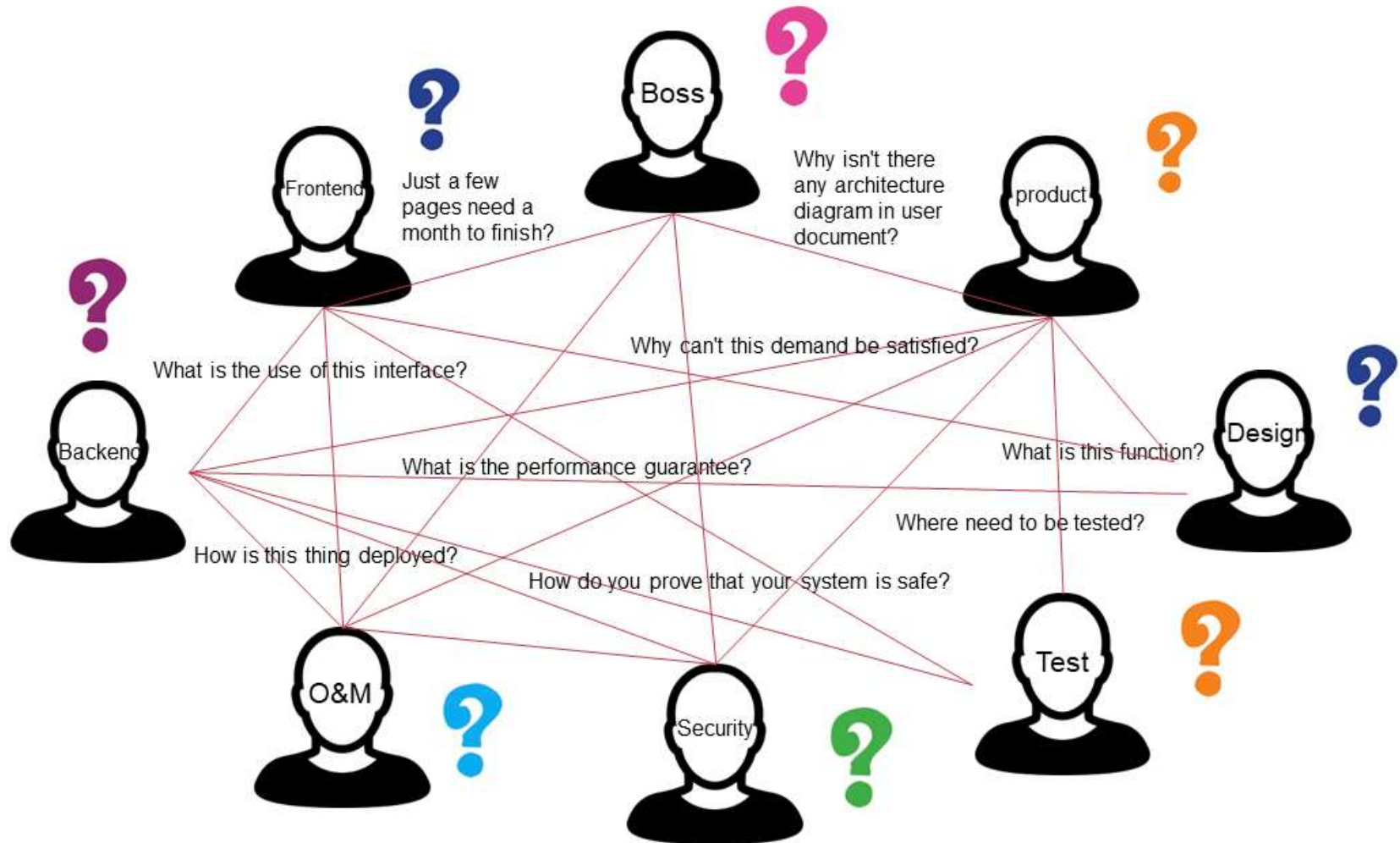
**Arhitektuur:** kuidas  
süsteem üldjoontes  
peaks olema ehitatud

Nõuded: mis on  
olulised asjad tegelikes  
“äriprotsessides”, mida  
peab arvestama

Kasutajaliidese ja  
tehnoloogiliste  
detailide disain,  
arendus, testimine



# Suurem pilt: vaja tekitada hea suhtlus ja arusaamine



- **Top down.** Reaalset “milleks seda vaja on” ehk ärivajadusi ja “mida peab arvestama, ehk mismoodi hakatakse kasutama” teab paremini (a) juhtkond ja (b) tulevased reaalsed kasutajad. See toob kaasa:
- **Bottom up.** Kuidas teha, mis on reaalne teha, mis lisaküsimused tekivad jne teavad paremini arendajad.

Alati on hea mõte suunata vajalik info kõigile osalistele, ehk, hoida **kõik pidevalt aruteluprotsessis**. Mh, et arendaja teaks ja tunnetaks ärivajadusi ja nõudeid.

Üks viis seda teha on nn **agiilne arendamine**: teeme väikeste funktsionaalsuste kaupa ja näitame / arutame seda pidevalt juhtkonnaga ja tulevaste reaalsete kasutajatega.

# Tarkvara arhitektuur ei ole eriala või eraldi distsipliin

Wikipediast: opinions vary as to the scope of software architectures:

- **Macroscopic system structure:** this refers to architecture as a higher-level abstraction of a software system that consists of a collection of computational *components* together with *connectors* that describe the interaction between these components.
- **The important stuff**—whatever that is: this refers to the fact that software architects should concern themselves with those decisions that have high impact on the system and its stakeholders.
- **That which is fundamental to understanding** a system in its environment
- **Things that people perceive as hard to change:** since designing the architecture takes place at the beginning of a software system's lifecycle, the architect should focus on decisions that "have to" be right the first time. Following this line of thought, architectural design issues may become non-architectural once their irreversibility can be overcome.
- **A set of architectural design decisions:** software architecture should not be considered merely a set of models or structures, but should include the decisions that lead to these particular structures, and the rationale behind them.

There is no sharp distinction between software architecture versus design and requirements engineering .

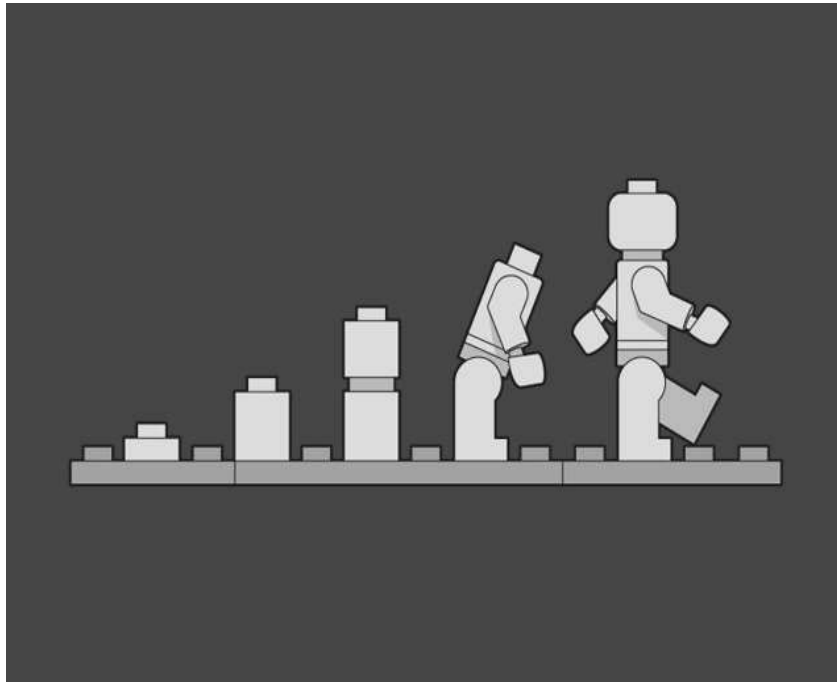


Olulisi punkte arusaamiseks

# Tarkvara ei ole lego

Tarkvara ei panda kokku “nagu lego blokkidest”:

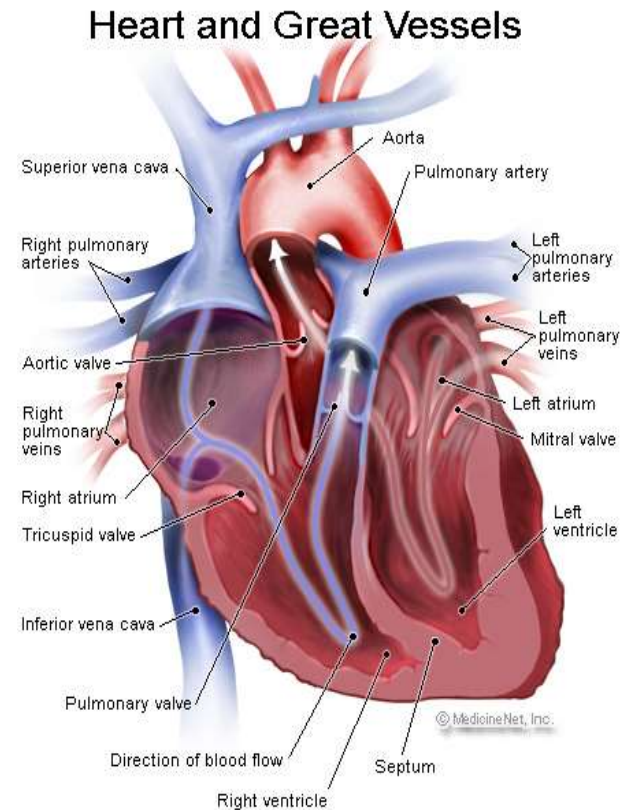
- Lego blokke saab ühendada väga lihtsalt, ja neil ei ole sisemist keerulist ehitust



# Tarkvara on sarnane bioloogiaga

Tarkvara ei panda kokku “nagu lego blokkidest”:  
ta sarnaneb enim **bioloogiaga**

- “Tarkvarablokid” on väga keerulised süsteemid, ja nende ühendamise tähendab nõ tavapärast programmeerimist
- Tarkvara areneb olemasolevast tarkvarast arendajate abil evolutsiooni teel, sarnaselt bioloogiaga



# Arendaja analoog on arhitekt/ehitusinsener

---

Tarkvara lihtne analoogia ehitusega ei ole eriti pädev:

- Tarkvara on kõige keerulisem asi maailmas: tarkvarasüsteemidel on kõige rohkem osi
- Arendaja kirjutatud programm ongi tegelikult ehitusjoonis, mitte ehitus: ehitus (töötav tarkvara) tehakse kompilaatorite, linkurite jms abil programmist automaatselt.
- Programmeerija “analoog” on arhitekt/ehitusinsener.
- Tarkvara arenduses ei ole nõ lihtsaid “ehitusmehi” vaja.
- Täiendava tööjõu lisamine projekti käigus teeb projekti täitmise reeglina veel aeglasemaks.

# Arhitektuuri kriitiline eesmärk: lihtsus

---

Arendamise ja haldamise lihtsus on kõige tähtsam.

- Valikute juures on otsustav see, mida arendajad/haldajad kõige paremini oskavad kasutada.
- Lisatükkide kasutamist ilma selge vajaduseta tuleks vältida.
- Abstraktsioonid tilguvad läbi.
- Ei ole olemas iga juhu jaoks sobilikke tehnoloogiaid/komponente

No silver bullet!



# Süsteemide komponendid

**Tarkvarasüsteemid** ehitatakse reeglina mitmesuguste komponentide kokkupaneku, s.t. kokkuprogrammeerimise teel, või teisiti öeldes: komponente kasutades.

Neid komponente võib klassifitseerida - näiteks - järgmisel viisil:

- Terviklikud lõppkasutaja-rakendusprogrammid
- Suured “valmiskomponendid”, näiteks andmebaasimootorid
- Raamistikud ehk frameworks
- Teegid ehk libraries

# Tarkvarasüsteemide komponendid

---

## Terviklikud lõppkasutaja-rakendusprogrammid,

Neid saab tihti juhtida ja mõne teise tarkvarapaketiga programmiselt siduda, a la:

- Tekstitöötlus, näiteks Word või LibreOffice
- Tabelarvutus, näiteks Excel
- Raamatupidamise tarkvara
- Arendustarkvara, näiteks Eclipse või vscode
- Veebibrauser



# Tarkvarasüsteemide komponendid

## Suured valmiskomponendid

Andmebaasiserverid, www-serverid, mailiserverid, graafikaserverid nagu X11, operatsioonisüsteem ise jne

Need programmid on tehtud eeskätt selleks, et lõppkasutaja jaoks loodud rakenduste tegemist hõlbustada. Näitena andmebaasi väljakutsumine Pythonist:

```
.....
cursor=con.cursor()
sqlstr= """  select clientid, clientname
              from clientbase
              where clientname like '%Jaan%' """
cur.execute(sqlstr)
results=cur.fetchall()
for i in results
    print "id: ",i[0], " name: ",i[1]
.....
```

## Raamistikud ehk frameworks

Edasiarendamiseks ja ümbertegemiseks mõeldud terviklikud näiterakendused, levinud eeskätt „tüüpiliste“ andmebaasi-kesksete veebirakenduste jaoks. Igast hästi läbimõeldud rakendusest võib saada selline näiterakendus. Populaarseid raamistikke:

- Java Spring
- Ruby on Rails
- PHP Zend framework
- Python Django
- Javascript Angular
- Microsoft .NET

Raamistik ei lase arendajal vabalt valida, kuidas süsteem peaks töötama, vaid pigem suunab teda täiendama etteantud näite-süsteemi, mille raamistiku arendaja on välja töötanud.

# Tarkvarasüsteemide komponendid

---

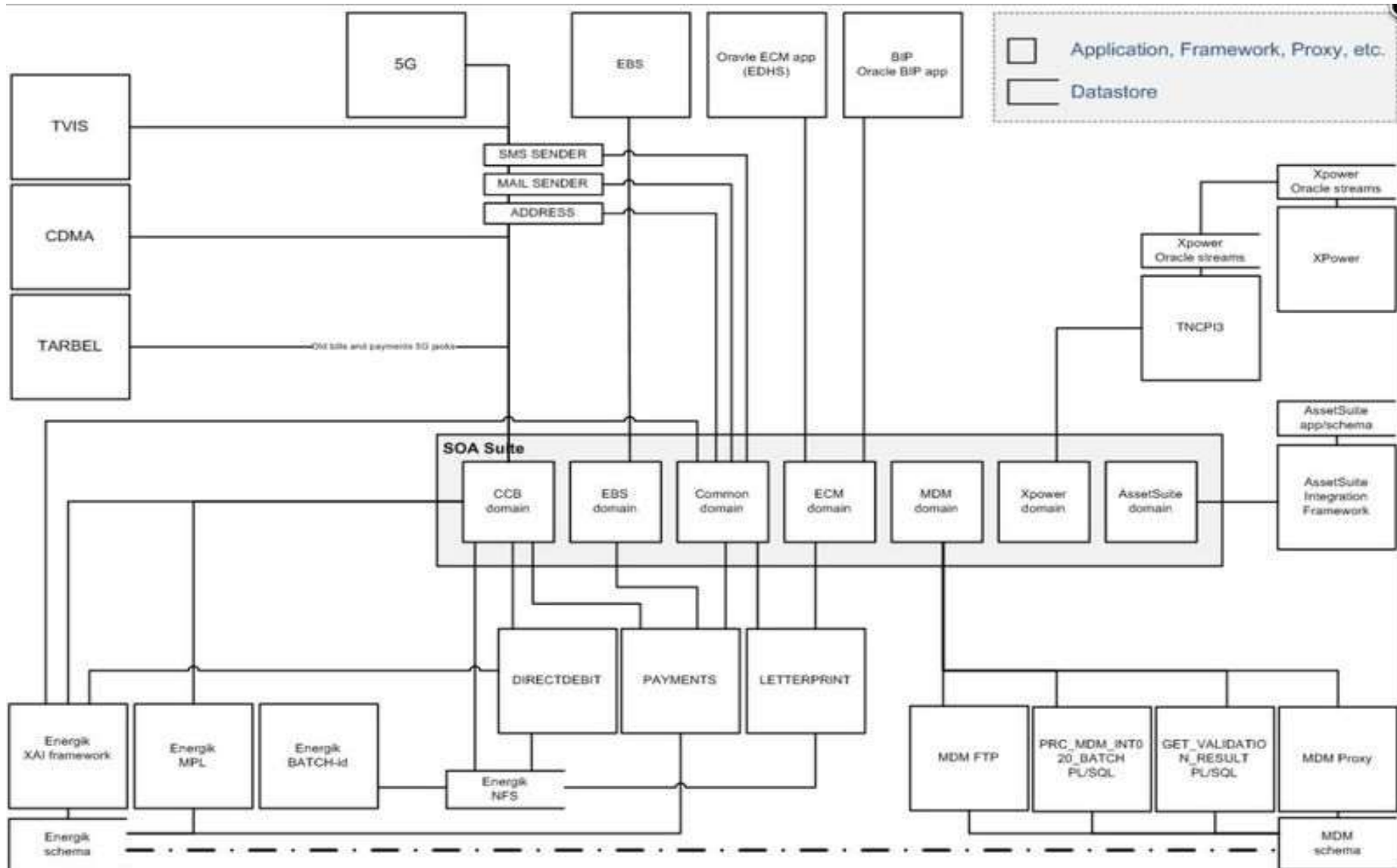
## Teegid ehk libraries

Konkreetseid, piiratud funktsioone realiseerivad väikesed komponendid ja nende komplektid. Mõned komplektid on hiigelsuured. Funktsioonide näiteid:

- Trükkimine
- Faili kirjutamine
- Võrguühenduse avamine
- Ringi, joone vms asja joonistamine

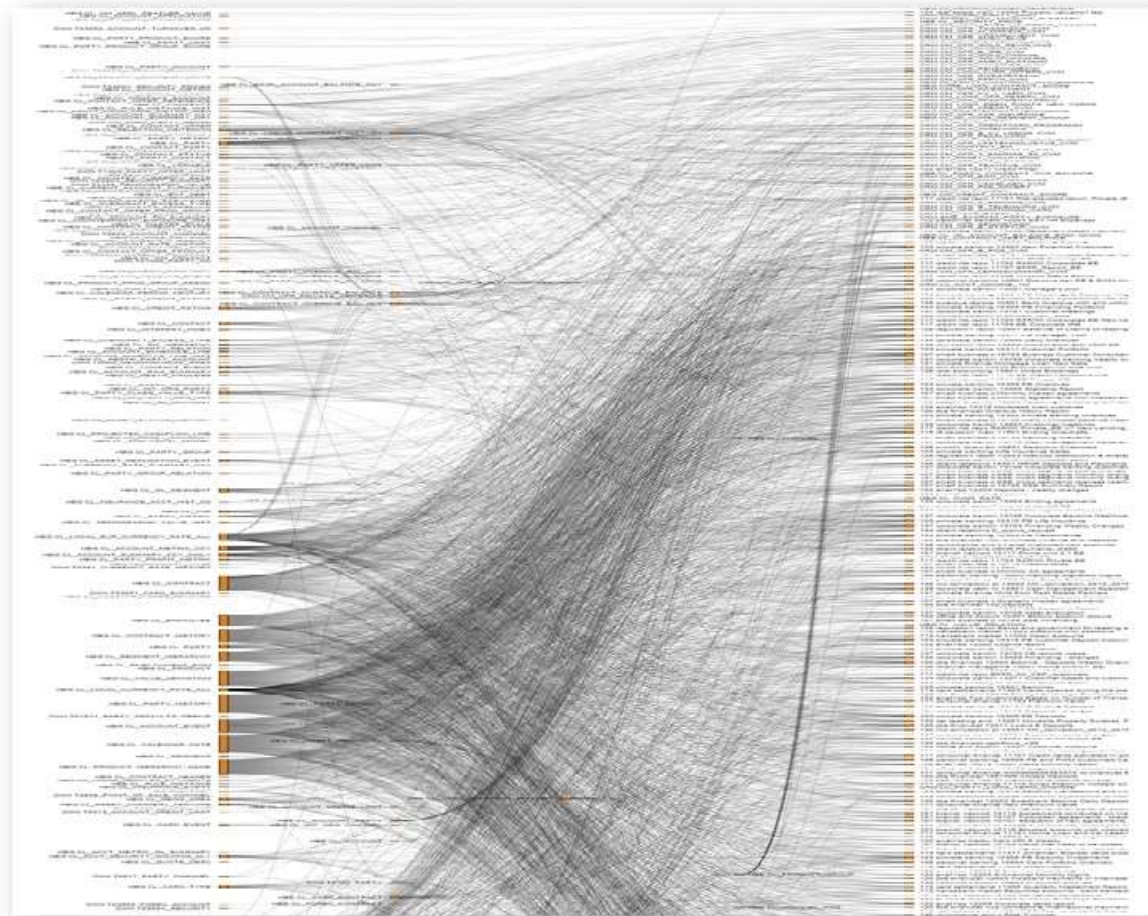
Selliseid komponente levitatakse enamasti komplekti kompilaatorite ja muude tarkvara-arendusvahenditega ning nad on tüüpiliselt kasutatavad ainult selle konkreetse programmeerimiskeele ja arendusvahendi koosseisus.

# Üks suur eesti infosüsteem: iga blokk on omaette keerukas süsteem oma andmebaasidega



# Andmete liikumise keerulised ahelad

## Andmete ja raportite seoste näide



# Meeldetuletus:

## algoritmilised keeled ja kirjelduskeeled

- **Algoritmilised (programeerimiskeeled)**

- C, C++, Basic, Java, Python, PHP, assembler ....

- **Kirjelduskeeled (spetsifitseerimiskeeled)**

- HTML, SQL, XML, RDF, TeX,...

- **Praktiline süntees:**

- algoritmilised keeled manipuleerivad kirjelduskeelee abil antavate objektidega

- näited: Javascript ja HTML., C/Java/Python/... ja SQL, ....

# Valdkonniti domineerivad programmeerimiskeeled

- Universaalseim, välja arvatud tippkiirust nõudvad või embedded või brauseri- või operatsioonisüsteemi-rakendused: **Java**
- Maksimaalset kiirust nõudvad rakendused, embedded ja süsteemprogrammeerimine: **C, C++**
- Andmetöötlus ja skriptid ilma kasutajaliideseta: **Python, Java, C, Go, Perl, Ruby**
- Windowsi kasutajaliidesega rakenduste programmeerimine: **C#, VisualBasic, C, (Java)**
- Maci ja iPhone programmeerimine: **Swift, Objective-C**
- Androidi programmeerimine: **Java, (C)**
- Veebibrauseri programmeerimine: **Javascript**
- Veebirakenduse programmeerimine: **PHP, Javascript, Python, Ruby, Java, Go, C , Perl ..**
- Spetsiifilised rakendused: vastavalt vajadusele

## **Programmi töötamiseks:**

- Kompilaator
- Interpretaator

## **Programmi kirjutamiseks:**

- Sobivad tekstiredaktorid
- Visuaalsed arendusvahendid

## **Suure hulga lähtekoodi halduseks:**

- Versioonikontroll (git, subversion, ...)
- Kompileerimissüsteemid (make, automake, Ant, Gradle, Webpack,...)
- Integratsioonisüsteemid (Jenkins, Travis CL, ...)

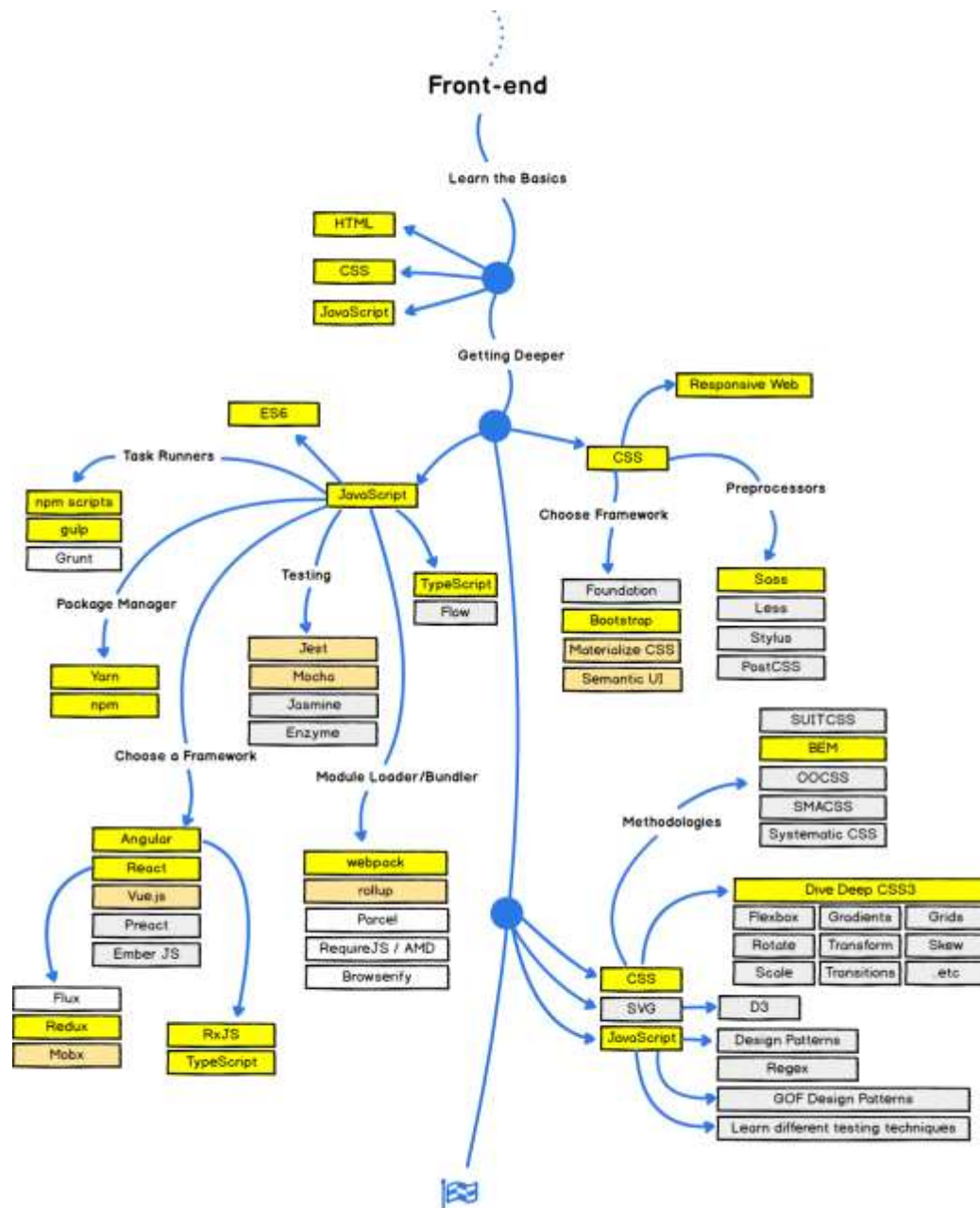
## **Testimiseks ja monitoorimiseks:**

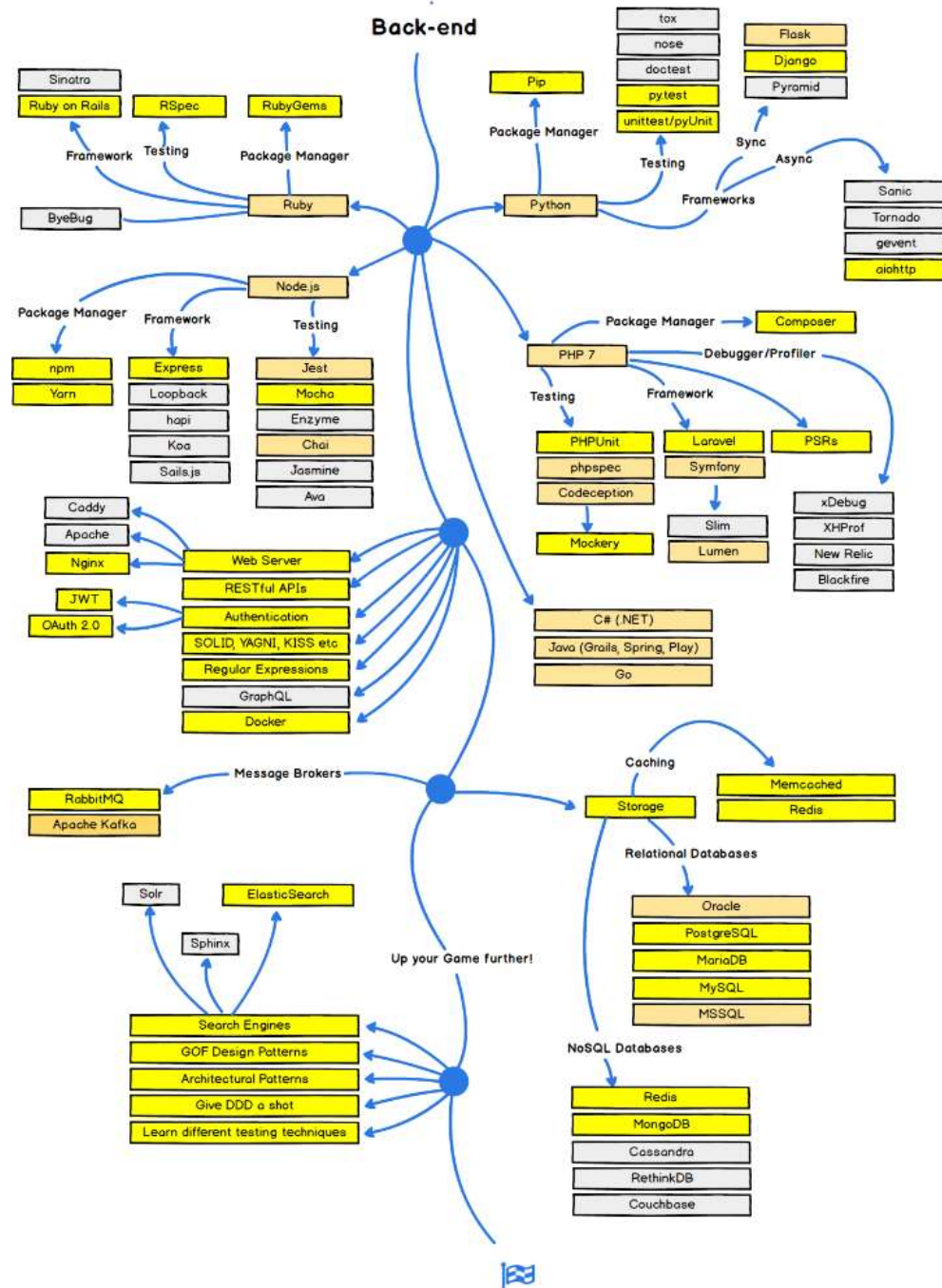
- Testimise abivahendid ja automaattestide süsteemid (Valgrind, Jtest, ...)
- Automaatmonitoorimise süsteemid



## **Veebitehnoloogiad: näide vahenditest**

**<https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d>**



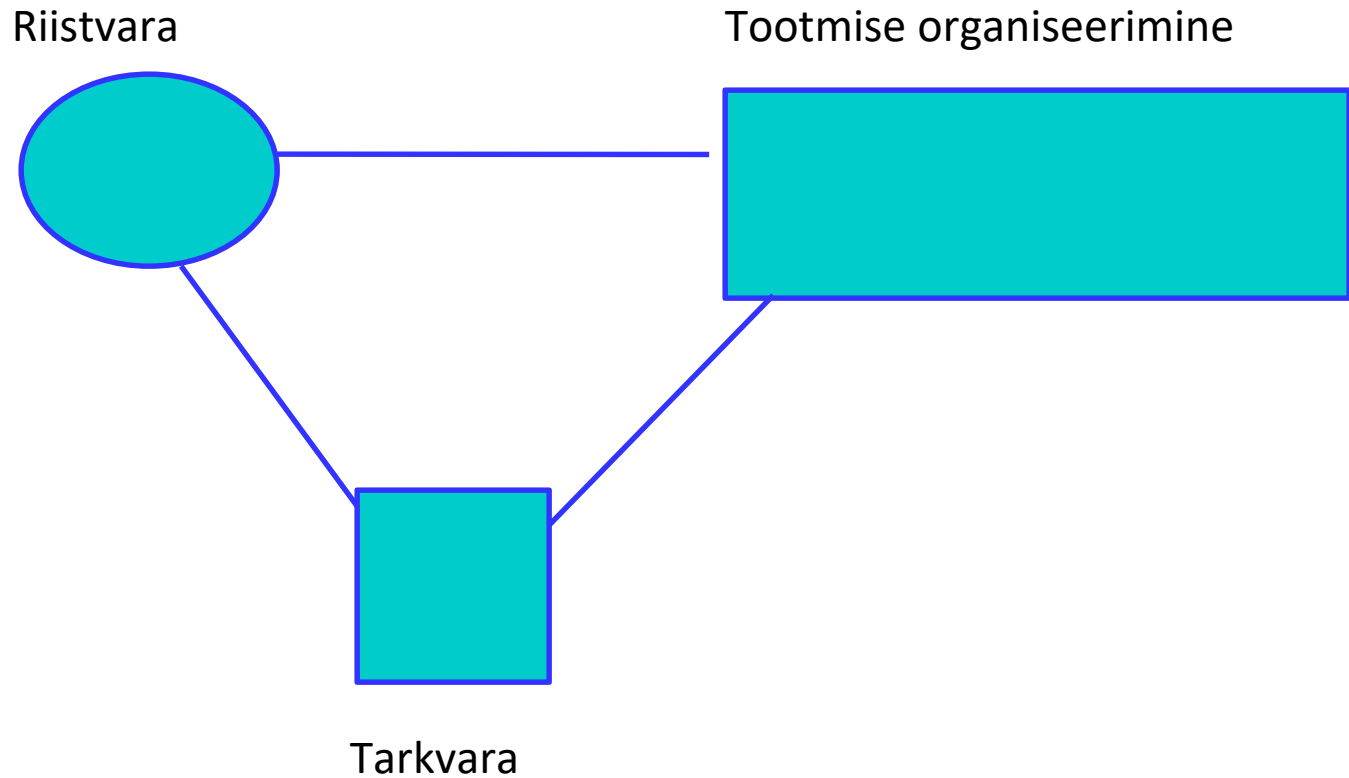


# Süsteemide sidumine

# IT süsteemide paradigmade areng

---

Ajalooline vaade kolmele osale:



# Pikaajalised arhitektuurifaasid: mis on olnud fookused

1945-1970

1970-1995

1995-2020?

A) **Riistvara** arhitektuurifaasid

Suurarvutid - Mikroarvutid - Võrgusüsteemid

B) **Tarkvaraplatvormide** arhitektuurifaasid:

Progekeeled -

teegid, arendusvahendid, komponendid -

komponentide sidumine

C) **Tootmise organiseerimise** arhitektuurifaasid:

suurfirma, avatud -

väikefirma, suletud -

vabad komponendid, sidumine, hooldus

# Priorities for software development

---

Three main consumers of time and effort:

- Understanding the business processes and needs.
- Understanding the exact contents of existing data.
- Writing code.

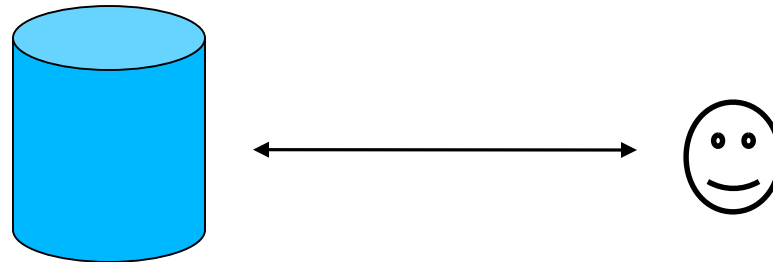
The second component - understanding existing data - is growing and will keep growing for foreseeable future.

**Why?**

# Architecture I

---

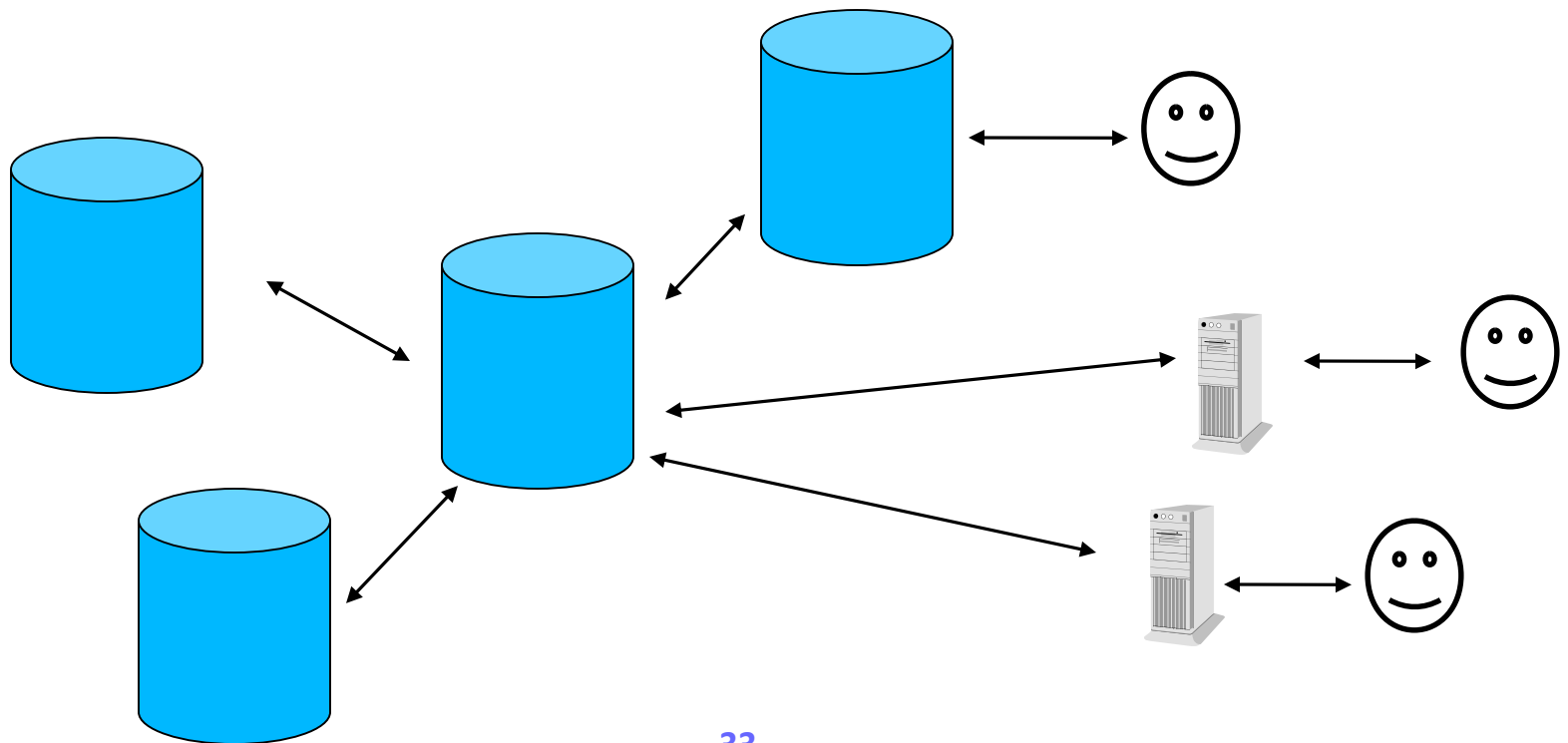
Compact standalone software





## Architecture II

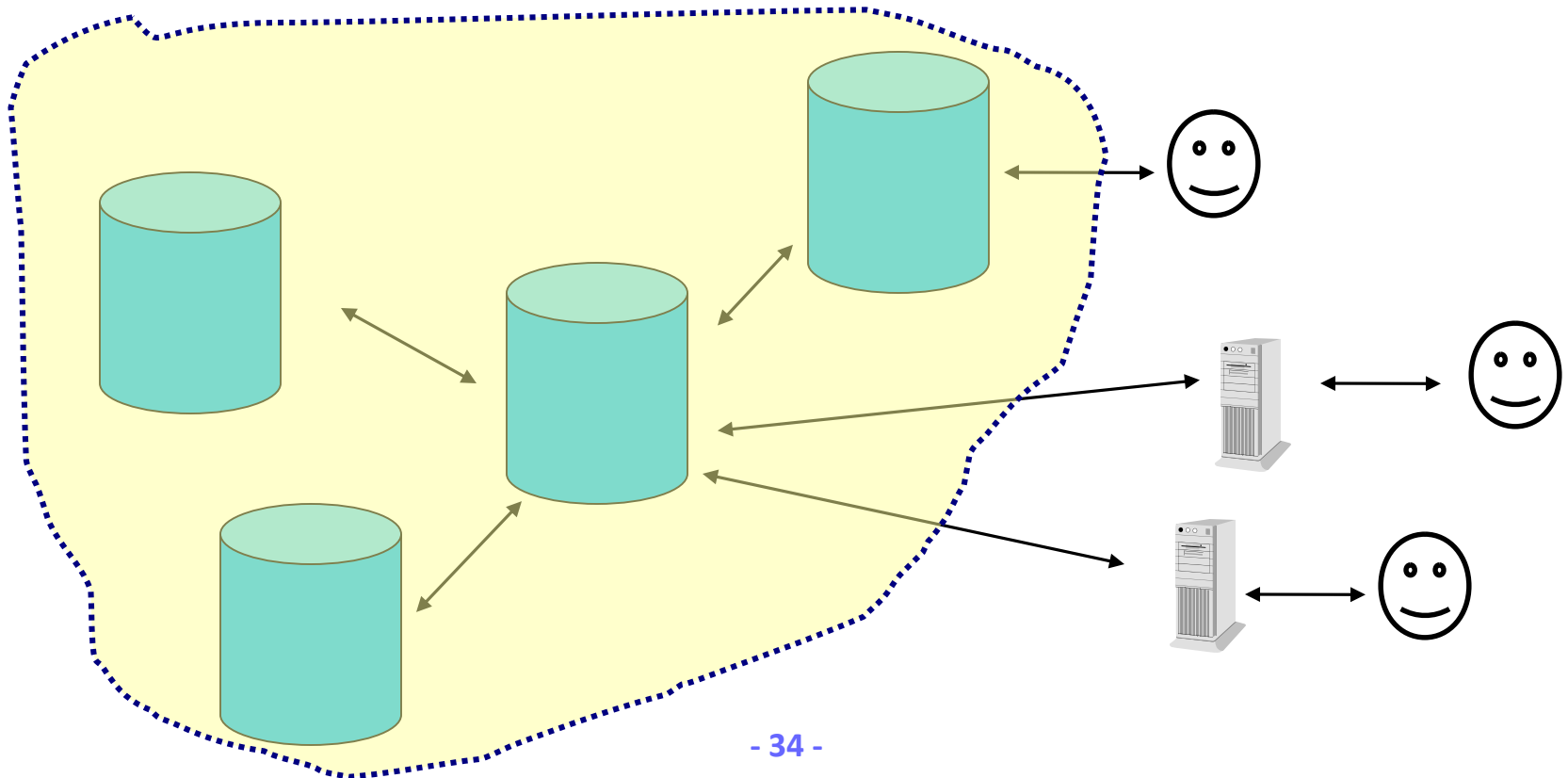
Classical client-server applications



## Architecture II : similarities to I

All applications controlled by the developer:

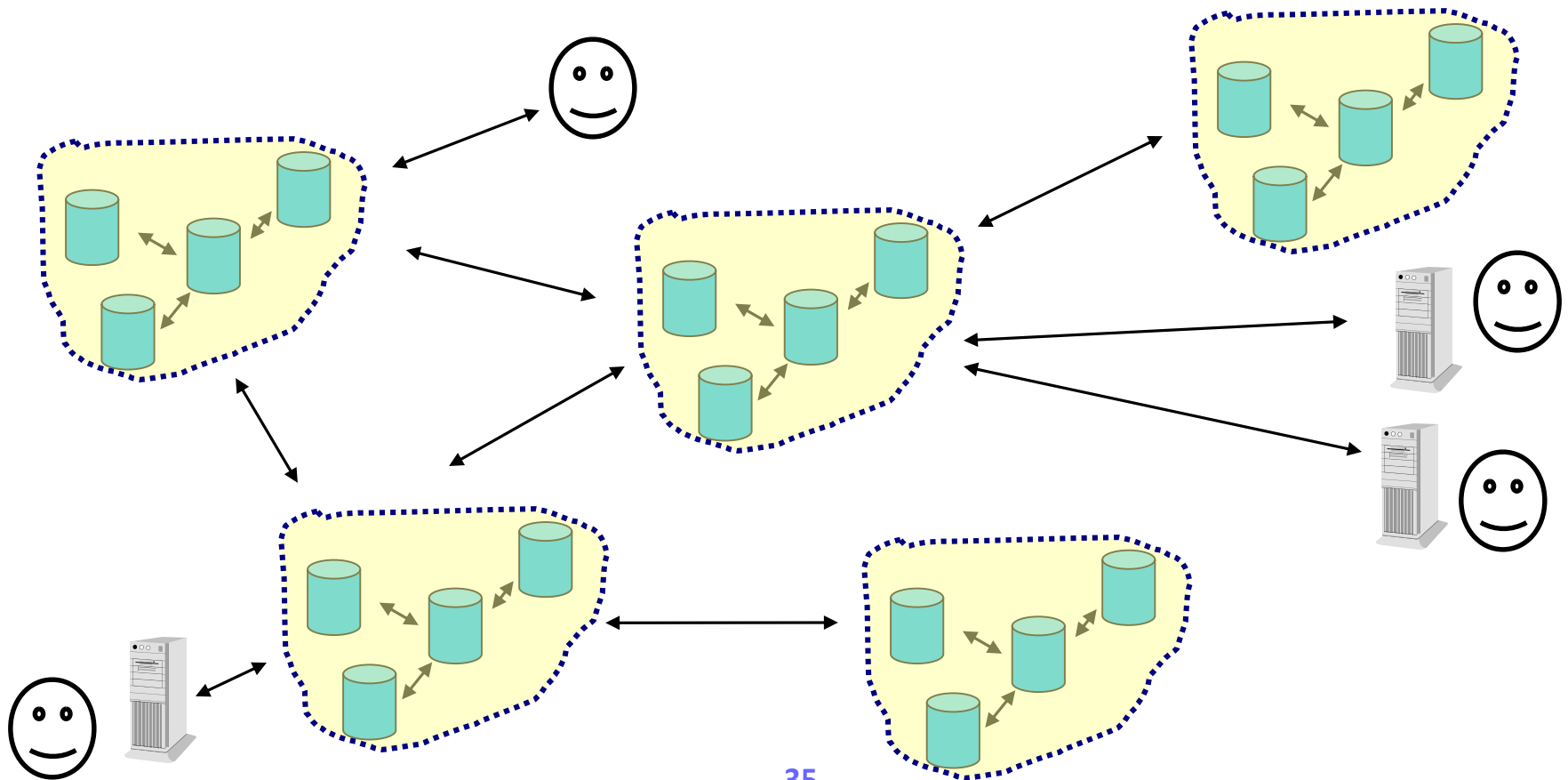
- located in one machine or
- distributed in the machines of the company/client



# Architecture III

Applications **not** controlled by the developer:

- you only have access to them
- system functions like a society of applications



Teistest arusaamine on põhiküsimus

# Military coalition exmple

---

Context: Yugoslavia, Iraq, Afghanistan ...

- Warfighting integration.
- We're basically tasked to close the seams in the kill chain.
- The „kill chain“ is a euphemism for the process by which we
  - identify targets
  - find them
  - fix
  - track
  - target
  - execute
  - check results



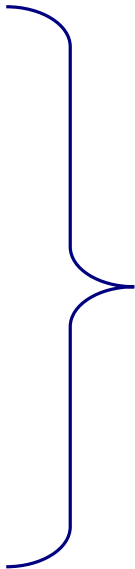
# Military coalition example

---

Context: Yugoslavia, Iraq, Afghanistan ...

- Warfighting integration.
- We're basically tasked to close the seams in the kill chain.
- The „kill chain“ is a euphemism for the process by which we

- identify targets
- find them
- fix it
- track it
- target
- execute
- check results



Every link is a separate country with its own information systems

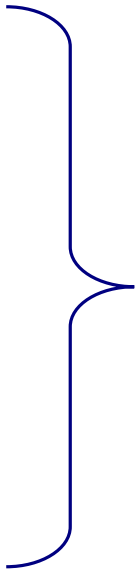
# Military coalition example

---

Context: Yugoslavia, Iraq, Afghanistan ...

- Warfighting integration.
- We're basically tasked to close the seams in the kill chain.
- The „kill chain“ is a euphemism for the process by which we

- identify targets
- find them
- fix it
- track it
- target
- execute
- check results



Every link is a separate country with its own information systems

Countries in the chain are often exchanged or replaced!

## Main problems for the developer

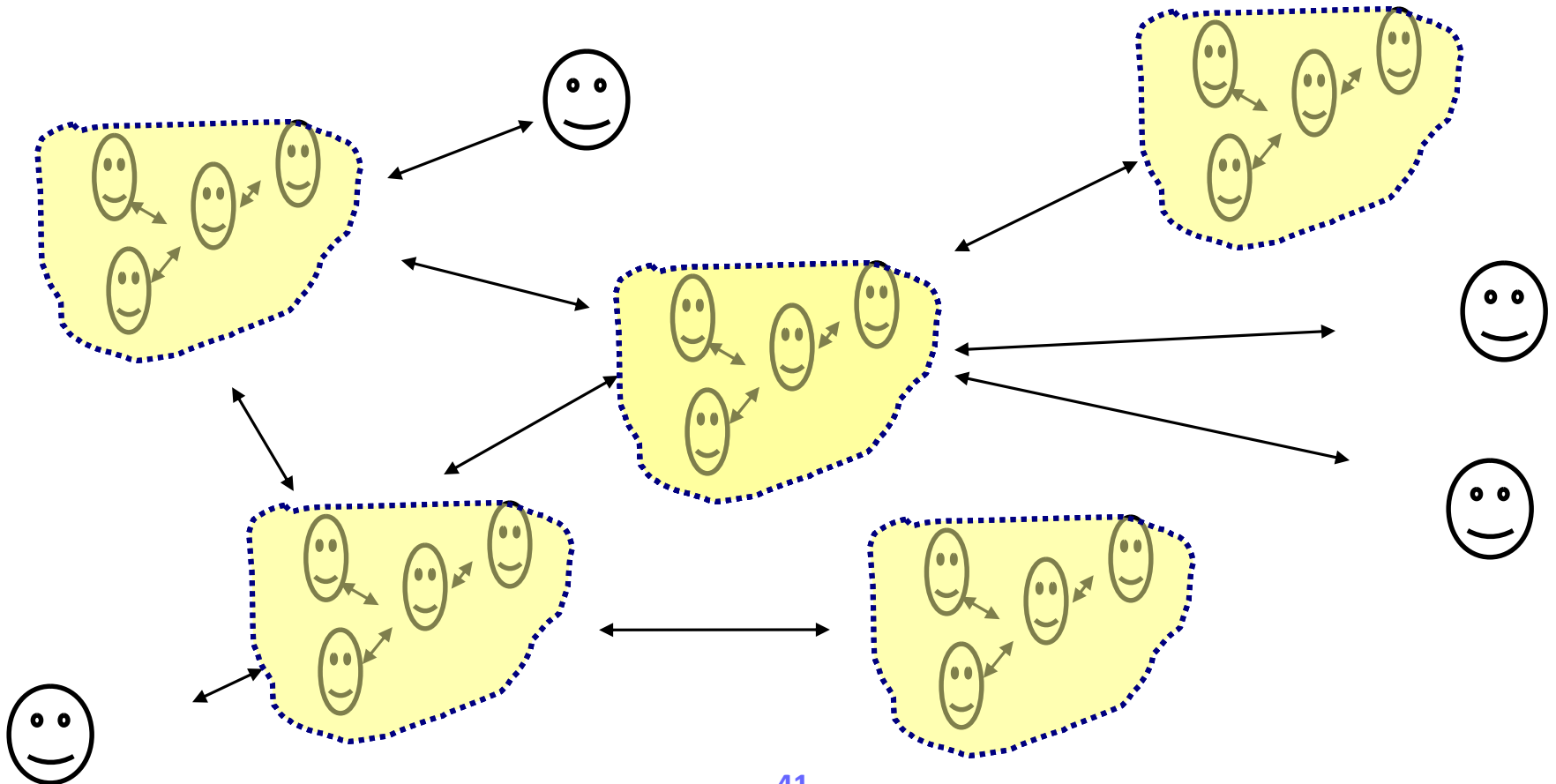
---

- **Get access** to data from the foreign system
- **Understand** what does this data exactly mean
- **Convert** foreign data to data structures of our own app and vice versa



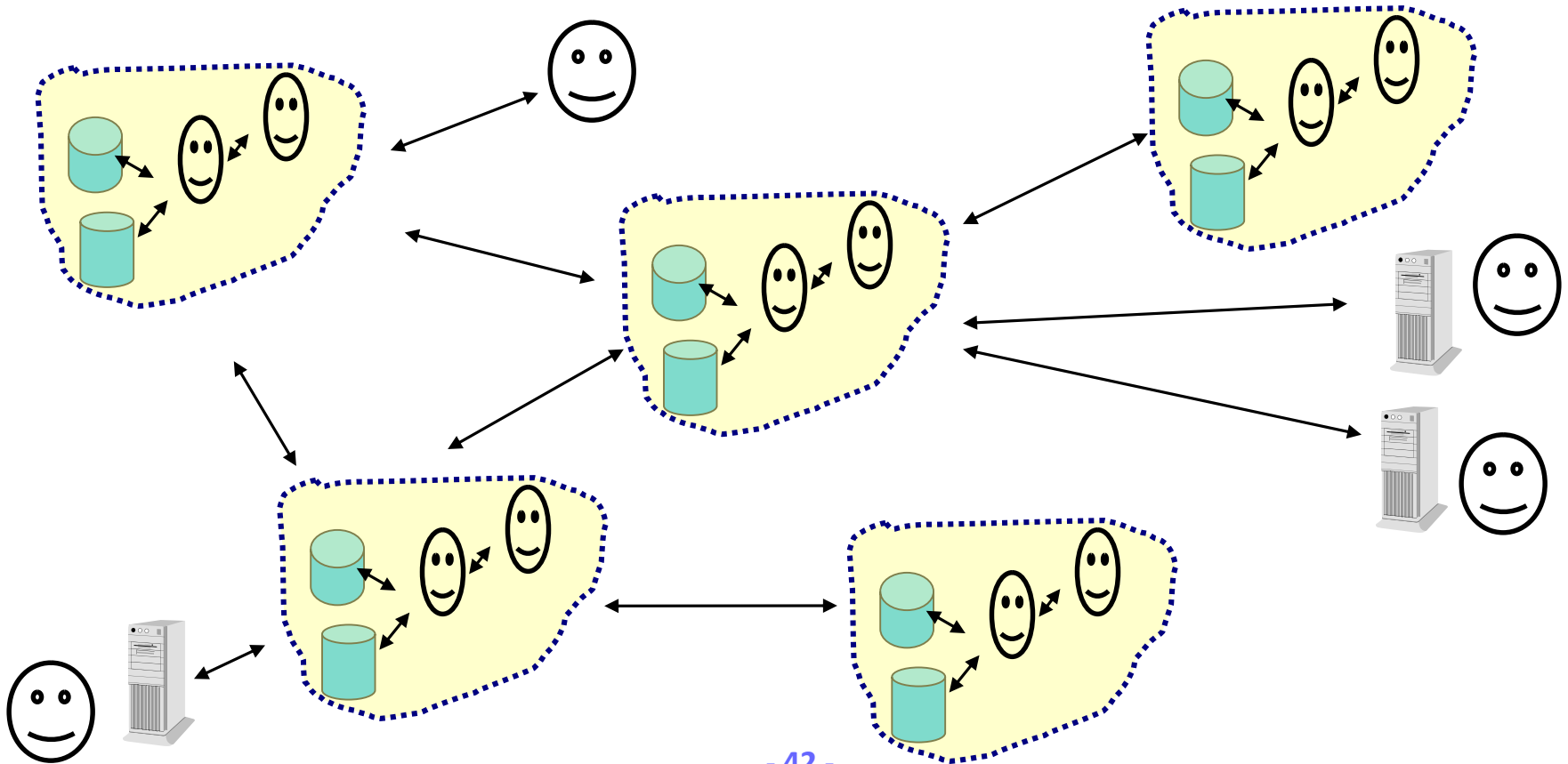
# Getting access and understanding?

Social networking. Connected groups of people:



# Architecture III revisited

Social groups consist of both people and apps



# Philosophy: society and people

---

- We could think that we do have AI already in some sense:  
society is a large animal  
with an intelligence of its own.
- **people -> society** is similar to  
**cells -> animal**



# Tarkvara litsentsid

## Tarkvara litsentsid

---

- Litsents on lihtsalt ühepoolne „võta-või-jäta“ leping, et kuidas tarkvara tohib kasutada.
- Kommerts litsentsid on väga mitmekesised: igaüks kirjutab endale sobiva.
- Vabavara on vahel dual-licenced: valid, kas kasutad piirangutega vabavarana või teistsuguste piirangutega kommerts litsentsiga.
- Vabavaralitsentside hulgas on mitmed rohkem/vähem vabad kategooriad.

Vaata ka:

<https://www.gnu.org/licenses/license-list.html>

## Levinumad kommerts litsentside tüübid

- **Single-user license** - The software is licensed for a single user and often a single computer.
- **Multi-user license** - This license allows you to install a program onto multiple computers used by multiple users. Typically this may be a set number of users. For example, a five user multi-user license allows up to five people to use the program.
- **Site license** - A program can be installed on an unlimited amount of computers, as long as they're at the location of the site license. Site licenses are usually for schools and businesses.

NB!

Iga autor võib litsentsi panna misiganes tingimusi.

Suurte / keeruliste süsteemide puhul tarkvara tihti ise ei kontrolli tingimuste täitmist.

## ■ Mis on vabatarkvara:

- tasuta?
- lahtise koodiga?
- edasimüügiõigusega?
- copyright?

## ■ Ajalugu, eesmärgid, perspektiivid

- mis on olnud?
- mis on tulemas?
- äriidee?
- miks vabavara tehakse?
- kus on vabavara mõistlik kasutada?
- mida vabavara on endaga kaasa toonud?

# GNU ideoloogia:

---

- vabadus: primaarne on tarkvara vabadus, sekundaarne tasuta kättesaadavus (free as in free speech, not as in free beer)
- ausus: ausam on kasutada vabavara kui piraatkopeerida
- teadmiste vabadus: teadmised, tarkvara on loomu poolest vaba - teadmiste ja tarkvara kopeerimine laiendab ühiskonna majanduslikku võimsust, kaotajaid (rumalamaks jääjaid) pole
- raha saab teenida ka tarkvara
  - toetades ja installeerides ja levitades
  - tellitud täiustusi ja modifikatsioone ehitades
  - ehitades sellist vabavara, mida klient soovib
- motivatsioon senistes kogustes tarkvara luua väheneb?
  - vastuargument: programmeerimine on põnev
  - vastuargument: ehk polegi meil nii palju kallilt makstud programmiste vaja?
  - vastuargument: vabanenud programmeerijate kaader saab teha uusi, senisest keerulisemat softi, mille jaoks seni aega ega ressursi ei jätkunud



- Vaatame eraldi GPL, LGPL ja muid vabavara-litsentse